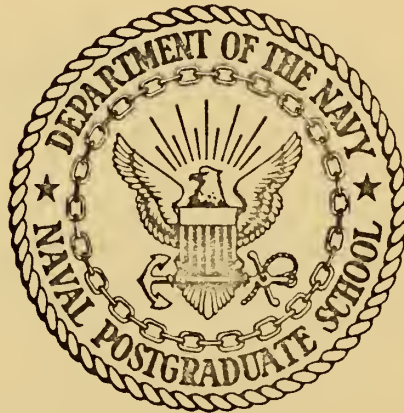


STEPS TOWARD A PL360-BASED COMPILER
GENERATOR FOR THE IBM 360 COMPUTER

Robert Charles Blanchard

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

STEPS TOWARD A PL360-BASED COMPILER
GENERATOR FOR THE IBM 360 COMPUTER

Robert Charles Blanchard

Thesis Advisor

G. A. Kildall/G. E. Heidorn

June 1972

Approved for public release; distribution unlimited.

Steps Toward a PL360-Based Compiler
Generator for the IBM 360 Computer

by

Robert Charles Blanchard
Lieutenant, United States Navy
B.S., Purdue University, 1967

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1972

ABSTRACT

The documentation of a complete proto-compiler consisting of a syntax checker and an OS/360 operating system interface for the IBM System/360 computers is presented. The system constitutes the foundation of a translator writing system based on the language PL360 and on the SLR(k) parsing algorithm. PL360 provides all the facilities of a symbolic machine language but displays an ALGOL-like structure for improved readability and programming ease. SLR(k) parsers have been shown to be superior to those constructed using precedence techniques with regard to the class of acceptable grammars and speed of operation.

TABLE OF CONTENTS

I.	INTRODUCTION -----	4
II.	BACKGROUND -----	6
	A. THE META PI SYSTEM -----	6
	B. THE XPL COMPILER GENERATOR SYSTEM -----	9
III.	DESCRIPTION OF THE PL360 COMPILER GENERATOR -----	12
	A. PL360 -----	12
	B. THE PARSING ALGORITHM -----	14
	C. THE PROTO-COMPILER -----	16
	D. OPERATING SYSTEM INTERFACE -----	16
IV.	CONCLUSIONS -----	19
	APPENDIX A. PROTO-COMPILER LISTING -----	20
	APPENDIX B. SYNTAX ANALYZER OUTPUT -----	30
	APPENDIX C. OS/360 OPERATING SYSTEM INTERFACE -----	32
	APPENDIX D. JOB CONTROL LANGUAGE -----	34
	BIBLIOGRAPHY -----	35
	INITIAL DISTRIBUTION LIST -----	36
	FORM DD 1437 -----	37
	KEYWORDS -----	38

I. INTRODUCTION

The computer solution to a problem is usually divided into two phases: translation of the source language and execution of the translated program. The translation process is a mapping of sentences from a source language to a target language while maintaining semantic equivalence. The target language is usually a sequence of machine instructions which directs the machine in the solution of the problem. The task of writing a program (compiler, assembler) to perform this translation process is generally long and difficult; thus, it has been the concern of researchers to automate as much of the compiler writer's task as possible through the use of translator writing systems (TWSs).

A TWS automates such functions as scanning text, analyzing syntax, synthesizing code, and interacting with an operating system in a general manner thereby allowing the compiler writer to concentrate on items unique to his translator.

The objective of the research reported herein was to develop a TWS based upon the language PL360 [Ref. 1] and to implement the system on the IBM 360/67 computer at the W. R. Church Computer Center, Naval Postgraduate School. This goal was not completely achieved since the system lacks a PL360 program to analyze a grammar and produce corresponding tables required by the SLR(1) [Ref. 2] parsing algorithm.

However, the basis of a TWS has been developed consisting of a syntax checker upon which the compiler writer may build, and an OS/360 operating system interface for the IBM System/360.

The next section of this report contains a review of two translator writing systems and is intended to provide the reader with some helpful background information. A description of the PL360 compiler generator is presented in the final section.

II. BACKGROUND

This section explores some concepts and principles of TWSs by reviewing two research efforts. An excellent paper by Feldman and Gries [Ref. 3] contains a critical survey many such efforts. The first system reported herein is that of O'Neil [Ref. 4] and the second is that of McKeeman, et al. [Ref. 5]. These were chosen because they are representative of the two general classes of parsing algorithms: goal oriented or top-down methods and bottom-up methods.

A. THE META PI SYSTEM

The META PI system is generally based upon a model known as META II developed by Schorre [Ref. 6] and his associates at UCLA. It is a syntax-directed symbol processor which states the parsing and translation functions for a language in a set of BNF-like (Backus-Naur Form) rules. These rules include semantic operations within the syntax structure describing the language. The basic parsing algorithm is simple top-down, left-to-right, with backup.

META PI statements contain three types of elements:

1. Syntactic elements which are used to generate the syntax checker of the user's compiler.
2. Semantic elements which affect code synthesis in the user's compiler.
3. META syntactic elements which enable the user's compiler to resolve possible ambiguities.

The user produces a compiler by combining these three elements into input statements.

The general form for a statement is:

LABEL := expression

The identifier to the left of the symbol pair "==" is defined by the right-hand expression. For example, the definition of IDENT is written:

IDENT := LETTER\$(LETTER/DIGIT)

The operator "\$" is a prefix iteration operator and indicates that whatever follows may be repeated any number of times (possibly zero). The META PI symbol "/" and the BNF symbol "|" are equivalent. Hence, the above rule would be interpreted as: "an identifier is a letter followed by any number of letters or digits." Note by the above example that unlike BNF, nonterminal symbols are not enclosed in broken brackets. Terminal symbols are preceded and followed by ":", and a "." indicates that a system symbol follows.

The elements that comprise META PI statements are explained in detail in Ref. 4; however, their use is illustrated by two examples below.

EXAMPLE 1:

If a user wished to permit multiple FORTRAN statements on one line, he might issue the following command:

USERCC := STAT.NOP(.CLAMP)\$STAT

where STAT identifies the definition of a FORTRAN statement. The semantic command ".NOP(...)" produces the effect of the semantic operation ".CLAMP" and has no effect on the compiler. It is included only to complete the general form of a semantic function, which is

semantic-command(semantic-operations)

The operation ".CLAMP" directs the compiler to suppress backup in the event of an exit to an error routine. Recall that "\$STAT" allows any number of statements to follow.

EXAMPLE 2:

To define a read statement equivalent to the BNF format

`<read statement> ::= READ <read list>`

the user might issue the following META PI command:

`READ := :READ: RID $(,: RID) :::`

where RID identifies the definition of a read-list element. Note that the word "READ", the comma, and the semi-colon will be recognized as terminal symbols because they are preceded and followed by the colon mark. A syntactically correct read statement could thus be

`READ <read-list element> , <read-list element> ;`

META PI generates an encoding of the rule in the user's compiler and references it by the unique identifier. When recognition of the identifier is established as a goal at compile time, the generated code is called and one of three condition codes is returned:

1. True, the scanned input satisfies the rule.
2. False, it does not.
3. Syntax error.

These three conditions describe the state of the top-down parsing algorithm at any given time. For example, assume the assignment statement

`DO1I = 1.5`

is the next input to be scanned. Note the similarity to the DO statement

`DO 1 I = 1,5`

which differs by the occurrence of a comma instead of a decimal point. Assume also that the current goal of the parser is STAT (statement) and that all statements are either DO statements or assignment

statements. If the parser first attempts to satisfy the subgoal of a DO statement, the condition code "false" is returned when the symbol "." is encountered. The subgoal of an assignment statement is then established and satisfied when the code "true" is returned. If the final parse attempt had also failed, the code "syntax error" would have been returned indicating that STAT could not be recognized.

Each recognition of an identifier causes a routine in the compiler to be executed and, in most cases, machine code to be generated.

Compilers produced by META PI are generally considered to be somewhat inefficient but have the advantages of ease of implementation and the ability to handle a large class of languages.

B. THE XPL COMPILER GENERATOR SYSTEM

In this section, the principles of McKeeman's compiler generator are discussed, concentrating on the parsing algorithm component. The system is explained in detail in Ref. 5, which is an excellent introduction to the construction of TWSs and a user's manual for the XPL programming language.

The parsing algorithm is a particular type of bottom-up parser. The distinguishing feature of the algorithm is that it does not use state-of-the-parse information, as top-down methods do; rather, it involves examining the canonical sentential form (each string in a canonical parse) to determine what unique parse step is applicable and then performs a substitution.

McKeeman's recognizer is a modification of Wirth's [Ref. 7] precedence concept. A wider class of grammars is acceptable since they are not restricted to simple precedence. The mixed-strategy precedence (MSP) algorithm uses a symbol pair predicate in its stacking

decision function and reverts to a bounded context of degree (2,1) predicate only in the case of a conflict. The production selection function is bounded context of degree (1,1); hence, the parser is called MSP (2,1;1,1). Since most triples of symbols cannot occur in a canonical parse, two symbols usually suffice to determine whether to accept the next symbol in the text and place it on the parse stack or to apply a parse step and reduce the top few symbols on the parse stack. Also, the number of different triples is so large (over 10,000 for XPL) that memory is wasted by tabulating all of them. Thus, McKeeman's concept of MSP is a compromise using Wirth's two-argument precedences whenever possible and switching to triples only when necessary.

The three major programs of the XPL compiler generator system are the syntax analyzer, which builds the tables required by the MSP algorithm; the proto-compiler with which the user can produce a compiler; and the XPL compiler which translates XPL to System/360 machine code.

The syntax analyzer is a program which accepts the BNF definition of a grammar, determines whether the grammar is, in fact, MSP (2,1;1,1), constructs parsing decision tables, and punches those tables on cards in the form of XPL declarations.

The proto-compiler uses the cards produced by the analyzer and functions as a syntax checker. The user may build on the proto-compiler by rewriting the code synthesis routine to implement the semantics of the new language to be compiled, and by altering the text-scanning routine to correctly interpret the terminal symbols of the new language. When this is done, each reduction in the syntax analysis causes the code synthesis procedure to be invoked. This procedure is provided the applicable production number so that the appropriate machine code can be generated.

The XPL compiler generator system allows the user to construct compilers for languages described by relatively small grammars with a minimum of effort.

III. DESCRIPTION OF THE PL360 COMPILER GENERATOR

It was decided to design a compiler generator system based on McKeeman's concepts but with two fundamental differences:

1. The system was to be written in the language PL360 to improve the performance of resulting compilers and to provide a more compatible interface to standard IBM software.
2. The parsing algorithm would be DeRemer's SLR(1) [Ref. 2] to increase the class of acceptable grammars and to improve computation time.

A. PL360

In 1968, Wirth published a formal description of PL360 [Ref. 1], a language designed specifically for the IBM System/360. A year later, a compiler written by Wirth, J. W. Wells, Jr., and E. Satterthwaite, Jr. was made available through the IBM Contributed Program Library [Ref. 8]. Several amendments to the original language definition were included with the documentation issued with the compiler. Further extensions and modifications to the language have recently been carried out, most notably by M. A. Malcolm [Ref. 9]. Malcolm's PL360 manual has incorporated all changes to the language definition and compiler description made to date.

PL360 is a language that provides all the facilities provided by System/360 Assembler Language yet exhibits an ALGOL-like syntax. It was designed to improve the readability of programs written to take advantage of specific capabilities and limitations of the System/360.

Such programs are defined by a set of BNF rules and semantic explanations given in Ref. 9.

Some characteristics of the language are illustrated by means of examples.

EXAMPLE 1:

```
PROCEDURE NEXTCHAR(R3);  
  
BEGIN IF R5 < 71 THEN R5 := R5 + 1 ELSE  
  
    BEGIN R0 := @CARD; READ; R5 := 0;  
  
    END;  
  
    IC(R0,CARD(R5));  
  
END
```

This procedure would be used to insert the next character of the input buffer into register R0 (bits 24-31). Register R3 contains the return address from this procedure so it is important not to alter its contents. Assume the identifier "CARD" has been previously declared to be a byte array of length 80. If register R5 (the column pointer) is not less than 71, then a new card is read by setting register R0 to the address of "CARD" and calling procedure READ. The column pointer is then initialized to zero and the first character of "CARD" is inserted into R0. If, however, R5 is less than 71, it is simply incremented and used as the new index.

EXAMPLE 2:

```
BEGIN INTEGER BUCKET;  
  
    IF FLAG THEN  
  
        BEGIN BUCKET := R0; R0 := R1; R1 := R2;  
  
            R2 := BUCKET;  
  
        END ELSE
```



```

        BEGIN BUCKET := R2; R2 := R1; R1 := RO;

        RO := BUCKET;

        END;

        RESET(FLAG);

    END

```

This block exchanges the contents of registers RO, R1, and R2 in a way which depends on the value of FLAG (assume FLAG has been declared to be a byte variable). If FLAG is true (i.e. if FLAG = #FF) then the first statement is executed and the contents of the registers are shifted left with R2 being assigned the value of RO. If FLAG is not true then the second statement is executed. The integer variable "BUCKET" is used as a temporary storage location. Upon completion of the IF statement, FLAG is set to #00 by a function call on RESET.

Note the block structure of the language with its attendant scope of variables. Note also the register manipulations and symbolic machine instructions (IC and RESET) which characterize assembler language.

B. THE PARSING ALGORITHM

DeRemer defines a class of context-free grammars called "Simple LR(k)" or SLR(k) which includes the simple precedence grammars as proper subsets. A method for constructing parsers for SLR(k) grammars is shown in Ref. 2; they have been implemented by DeRemer and have proven to be superior to corresponding MSP parsers both in the speed of parser construction and in the size and speed of the resulting parsers.

For the stacking decision, the MSP algorithm uses at most the top two symbols on the parse stack and the next symbol from the input text. SLR(k) parsers make the stacking decision based on all the symbols in

the parse stack plus k more from the input text. This is accomplished by restructuring the stack and saving state-of-the-parse information. The operation of the parser will be illustrated by example.

Consider a sample grammar:

$$G ::= E$$

$$E ::= E + T \mid T$$

$$T ::= (E) \mid x$$

The finite state machine represented in Fig. 1 parses sentences generated by the sample grammar. The algorithm is started in state 0 and passes through a series of states until reaching a state with no successor. The indicated rule is applied and the parser is restarted in state 0. The algorithm terminates upon reaching state 2 and an end-of-file mark is encountered. For example, when in state 1 and the symbol "x" is encountered, apply the reduction $T \rightarrow x$ and restart; when in

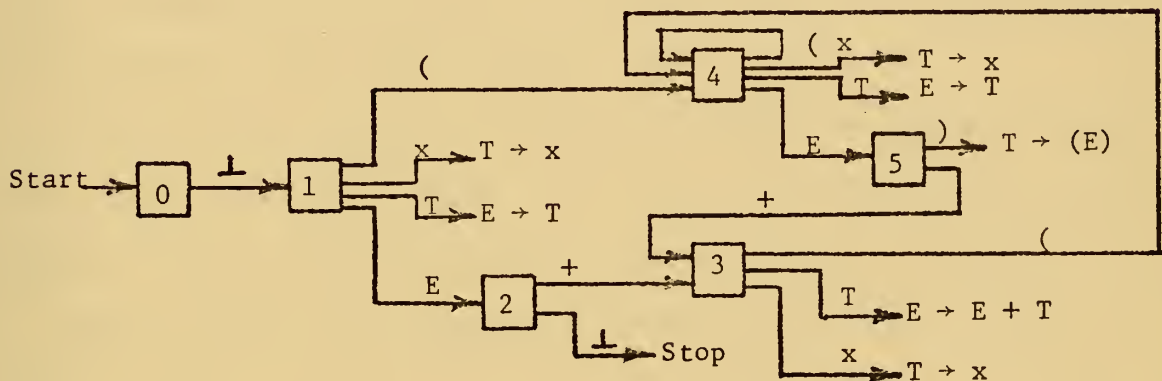


Fig. 1 Finite State Machine for the Sample Grammar

state 3 and the symbol "(" is encountered, stack the symbol and enter state 4.

The SLR(1) parsing algorithm has been implemented in one of the procedures of the proto-compiler.

C. THE PROTO-COMPILER

The program called "PROTOCOL" forms the basis of the PL360 compiler generator system (see Appendix A for the program listing). It is patterned after McKeeman's proto-compiler and has the same function and basic structure.

PROTOCOL uses SLR(1) parsing tables obtained by manually translating the XPL declarations produced by DeRemer's syntax analyzer [Ref. 10] into equivalent PL360 declarations. The output from DeRemer's program is listed in Appendix B. The translated tables are referenced by the algorithm contained in procedure ANALYZE to implement the finite state machine of a simple grammar.

ANALYZE calls on procedure PUSHANDREAD or procedure SYNTHESIZE based on a decision to stack the current symbol and read a new one or to make a reduction and perform the required semantic operations. Since code synthesis is not a function of PROTOCOL, SYNTHESIZE exists only to maintain flow of control and indicate that its presence would be required in a full-scale compiler.

Procedure SCAN, called from either PUSHANDREAD or ANALYZE, interprets characters in the card buffer. Upon reaching the end of a card image, a call on procedure GETCARD causes a new card to be read.

The only other major procedure is ERROR which is called from a number of other routines; it accounts for syntax errors and prints diagnostic messages.

D. OPERATING SYSTEM INTERFACE

Assuming that the object program resulting from a compilation of PROTOCOL is to be used as a compiler, the problem of communicating with an operating system arises. Since PL360 object programs do not

communicate directly with an operating system, an interface program must be separately assembled and merged with the compiler by the linkage editor.

The existing system uses such an interface (called "PLIO") in the OS/360 operating system environment (see Appendix C for the program listing). It has entry points to four subroutines which facilitate input and output operations. The names and specifications of these subroutines are listed below.

1. READ: transmission of a card image record to PROTOCOM; RO contains the address of an 80-byte buffer into which the record is to be moved; set condition code to 2 if end-of-file is encountered, otherwise set to 0.
2. WRITE: transmission of a line image record from PROTOCOM; RO contains the address of a 132-byte output record.
3. PAGE: insert the USASI control character "1" into the first position of the print buffer.
4. PUNCH: transmission of a card image from PROTOCOM; RO contains the address of an 80-byte output record.

Two additional subroutines would be required in PLIO if PROTOCOM is to be built into a compiler: a system initialization subroutine to decode any required parameter list, open required data sets, obtain free storage, and supply system identification. A system termination subroutine is also necessary to release free storage and close required data sets. These enable the operating system to properly identify and store the machine code produce by the compiler.

PROTOCOM uses the data sets described below and identified by their DDNAMES. All data sets are sequential with fixed block format.

1. SYSIN: This data set constitutes the input and consists of one or more source programs. The logical record length is 80 bytes.
2. SYSOUT: This data set contains the compiler output listing including diagnostic messages. The logical record length is 133 bytes.
3. SYSPUNCH: This data set contains object code (if any) produced by the compiler. The logical record length is 80 bytes.

At least one additional data set would be required in the event PROTOCOM is built into a compiler: a direct access data set to contain object code and closed with a disposition of REREAD for further processing, such as linkage editing.

The job control language required to compile, linkage edit, and execute a typical PL360 program is listed in Appendix D. It is assumed that the file with DSNAME S0938.PLLIB contains the PL360 compiler and its required interface routines.

The author believes PROTOCOM to be free of errors. The program, however, has not been subjected to rigorous debugging. Also, in the interest of readability and the lack of a pressing need, no attempt has been made to write the most efficient proto-compiler possible.

IV. CONCLUSIONS

It is concluded that the PL360 language is well suited to form the basis of a compiler generator system which is to be implemented on the IBM System/360. It is fairly successful in providing a tool which is superior to assembly code and in meeting the objectives of readability and writability. The language is not as easy to use as some other high-level compiler-writing languages, such as XPL. The execution speed, however, indicates it to be the superior language in systems and production programming applications.

It was stated earlier that the compiler generator system described in this paper has not been fully implemented in that it lacks a syntax analyzer. The method of manually translating XPL declarations produced by DeRemer's program into equivalent PL360 declarations is conceptually simple but tedious for interesting grammars. It would not be a difficult task to alter DeRemer's program to have it produce PL360 declarations directly. It is recommended that an SLR(1) analyzer written in PL360 be added to the system to provide a solution to this problem.

APPENDIX A

PROTO-COMPILER LISTING

```

$TITLE      PROTOCOM
BEGIN
COMMENT     THIS PROGRAM IS A PROTO-COMPILER WRITTEN IN PL360.

              THIS VERSION OF PROTOCOM IS A SYNTAX CHECKER FOR
              THE FOLLOWING GRAMMER:

<G> ::= <E>

<E> ::= <T>
      | <E> + <T>

<T> ::= <P>
      | <T> * <P>

<P> ::= X
      | Y
      | Z
;

COMMENT     ASSUME THE FOLLOWING CARDS WERE PUNCHED BY THE
              SLR(1) SYNTAX ANALYZER;

INTEGER NUMTERMINALS = 6;
INTEGER NUMNTS = 4;
INTEGER NUMSYMS = 11;

ARRAY 11 LONG REAL V = (
    #C5D9D9D6D9E2E8D4L, COMMENT "ERRORSYM";
    #00000000006D4F6DL, COMMENT " | ";
    #0000000000000004EL, COMMENT "+ ";
    #0000000000000005CL, COMMENT "* ";
    #000000000000000E7L, COMMENT "X";
    #000000000000000E8L, COMMENT "Y";
    #000000000000000E9L, COMMENT "Z";
    #000000000004CC76EL, COMMENT "<G>";
    #000000000004CC56EL, COMMENT "<E>";
    #000000000004CE36EL, COMMENT "<T>";
    #000000000004CD76EL); COMMENT "<P>";

COMMENT     THE DPDA HAS 8 READ STATES;

ARRAY 8 SHORT INTEGER READSTART = (#0000S, #0002S, #0009S,
    #000BS, #000DS, #000FS, #0016S, #001DS);

ARRAY 8 BYTE RDNUM = (#01X, #07X, #01X, #01X, #01X, #07X,
    #07X, #01X);

ARRAY 31 BYTE SYMLIST = (#01X, #00X, #00X, #01X, #02X, #03X,
    #04X, #05X, #06X, #01X, #00X, #02X, #00X, #03X, #00X,
    #00X, #01X, #02X, #03X, #04X, #05X, #06X, #00X, #01X,
    #02X, #03X, #04X, #05X, #06X, #03X, #00X);

ARRAY 31 SHORT INTEGER STATELIST = (#0101S, #06FFS, #06FFS,
    #06FFS, #06FFS, #06FFS, #0206S, #0207S, #0208S, #0209S,
    #06FFS, #0105S, #06FFS, #0106S, #06FFS, #06FFS, #06FFS,
    #06FFS, #06FFS, #0206S, #0207S, #0208S, #06FFS, #06FFS,
    #06FFS, #06FFS, #0206S, #0207S, #0208S, #0106S, #06FFS);

COMMENT     THE DPDA HAS 9 REDUCE STATES;

ARRAY 10 BYTE NUMTOPOP = (#00X, #00X, #00X, #02X, #00X, #02X,
    #00X, #00X, #00X, #02X);

```



```

ARRAY 10 SHORT INTEGER REDUCESUCC = (#0000S, #0002S, #0300S,
#0300S, #0500S, #0500S, #0501S, #0501S, #0501S, #0700S);
COMMENT THE DPDA HAS 3 LOOK-AHEAD STATES;
ARRAY 3 BYTE LASYMMNUM = (#00X, #00X, #00X);
ARRAY 3 SHORT INTEGER SUCCSTATE = (#0201S, #0202S, #0203S);
ARRAY 3 SHORT INTEGER FAILSTATE = (#0003S, #0004S, #0007S);
ARRAY 3 REAL LATABLE = (
#40000000R COMMENT <G>;,
#60000000R COMMENT <E>;,
#60000000R COMMENT <E>;);
COMMENT THE DPDA HAS 2 LOOK-BACK STATES;
ARRAY 2 BYTE LBSTART = (#00X, #02X);
ARRAY 2 BYTE LBNUM = (#01X, #01X);
ARRAY 4 SHORT INTEGER LBSTATE = (#0101S, #0000S, #0106S,
#0000S);
ARRAY 4 SHORT INTEGER RESUMESTATE = (#0301S, #0302S, #0205S,
#0204S);
COMMENT THE SYMBOLS ACCESSING THE STATES;
ARRAY 8 BYTE SYMBEFOREREAD = (#00X, #01X, #07X, #08X, #09X,
#02X, #03X, #09X);
ARRAY 3 BYTE SYMBEFORELA = (#08X, #09X, #09X);
COMMENT END OF CARDS PUNCHED BY THE SLR(1) SYNTAX ANALYZER;
INTEGER RESERVEDLIMIT = 0;
LONG REAL V8 SYN V(8); COMMENT SHOULD ALWAYS BE " | ";
ARRAY 30 BYTE ALPHABET = ("ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#");
COMMENT DECLARATIONS FOR THE SCANNER:
      TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE
      LAST SYMBOL SCANNED, CP IS THE POINTER TO THE LAST
      CHARACTER SCANNED IN THE CARD IMAGE, BCD IS THE
      LAST SYMBOL SCANNED;
COMMENT NUMBEVALUE CONTAINS THE NUMERIC VALUE OF THE LAST
      CONSTANT SCANNED;
INTEGER TOKEN = 1, PRODNUM, NUMBEVALUE, SP;
INTEGER REGISTER CP SYN R8;
LONG REAL BCD;
INTEGER BCDHIGH SYN BCD(0);
INTEGER BCDLOW SYN BCD(4);
ARRAY 80 BYTE CBUF; COMMENT CARD BUFFER;
COMMENT EXITFLAG IS USED TO INDICATE END OF COMPILING;
BYTE LISTFLAG, ENDIT, EXITFLAG = #00X;
COMMENT XR IS THE ERROR ROUTINE PARAMETER REGISTER;
INTEGER REGISTER XR SYN R5;
SHORT INTEGER ERRCOUNT = 0S, CARDCOUNT = 0S;
INTEGER EOFILE, NUMBER, IDENT, DIVIDE;
LONG REAL CONWORK; COMMENT USED TO CONVERT TO DECIMAL;
BYTE TRUE = #FFX, FALSE = #00X;
SHORT INTEGER PREVIOUSERROR, ERRLIMIT = 50S;
ARRAY 132 BYTE BLANK = 132(" ");
ARRAY 132 BYTE WBUF; COMMENT WRITE BUFFER;
INTEGER MASK = #000000FF;
INTEGER MASK7 = #00000007;
INTEGER MASKFFFF = #0000FFFF;
INTEGER BLANKMASK = #40404040;
INTEGER MASK1 = #00000001;

```



```

INTEGER MASKFF00 = #0000FF00;
LONG REAL VR3;
INTEGER VR3HI SYN VR3(0);
INTEGER VR3LOW SYN VR3(4);
ARRAY 256 BYTE CHARTYPE = 256(1);
ARRAY 256 BYTE NOTLETTERORDIGIT = 256(1);
ARRAY 256 INTEGER TX;
INTEGER TEXTLIMIT = 71; COMMENT SCAN TO CBUF(TEXTLIMIT);
ARRAY 10 BYTE NUMS = ("0123456789");

```

```

FUNCTION SETZONE(8,#96F0); COMMENT FUNCTION TO SET ZONE;
PROCEDURE ERROR(R4); COMMENT PRINTS AND ACCOUNTS FOR ALL
ERROR MESSAGES;

```

```

BEGIN INTEGER SAVE4;
SAVE4 := R4; RO := ERRCOUNT + 1; ERRCOUNT := RO;
IF RO > ERRLIMIT THEN GOTO X;
COMMENT IF LISTING IS SUPPRESSED, FORCE PRINTING OF
THE CARD BUFFER;
IF ¬LISTFLAG THEN
BEGIN
RO := CARDCOUNT + 1; CVD(RO,CONWORK);
UNPK(3,7,WBUF(45),CONWORK); SETZONE(WBUF(48));
MVC(79,WBUF(52),CBUF); RO := @WBUF; WRITE;
MVC(131,WBUF,BLANK);
END;
MVC(9,WBUF,"*** ERROR,");
CASE XR OF
BEGIN
NULL; COMMENT CASE 1 NOT USED;
BEGIN
R1 := @WBUF(CP+52); MVI("|",B1);
MVC(17,WBUF(11),"ILLEGAL CHARACTER:");
END;
BEGIN
MVC(15,WBUF(11),"STACK OVERFLOW. ");
SET(EXITFLAG);
END;
BEGIN
MVC(19,WBUF(11),"ILLEGAL SYMBOL PAIR:");
END;
BEGIN
MVC(24,WBUF(11),"PROGRAM ENDS PREMATURELY.");
SET(EXITFLAG);
END;
END;
RO := @WBUF; WRITE; MVC(131,WBUF,BLANK);
IF EXITFLAG THEN GOTO EXIT;
RO := ERRCOUNT;
IF RO > 1 THEN
BEGIN
MVC(34,WBUF,"*** LAST ERROR DETECTED ON LINE ");
RO := PREVIOUSERROR; CVD(RO,CONWORK);
UNPK(3,7,WBUF(33),CONWORK); SETZONE(WBUF(36));
MVC(4,WBUF(37),". ***"); RO := @WBUF; WRITE;
MVC(41,WBUF,BLANK);
END;
RO := CARDCOUNT; PREVIOUSERROR := RO;
RO := ERRCOUNT;
IF RO = ERRLIMIT THEN
BEGIN
MVC(20,WBUF,"*** TOO MANY ERRORS, ");
MVC(21,WBUF(20)," CHECKING ABORTED. ***");
END;
X: R4 := SAVE4;
END;

```

```

PROCEDURE GETCARD(R4);
COMMENT DOES ALL CARD READING AND LISTING;
BEGIN INTEGER SAVE4;
SAVE4 := R4; RO := @CBUF; READ;

```



```

IF ^= THEN COMMENT SIGNAL FOR EOF;
BEGIN
  SET(ENDIT); MVC(79,CBUF,BLANK);
  MVC(10,CBUF,"EOF;END;EOF");
END;
COMMENT CARDCOUNT PRINTED ON LISTING;
R0 := CARDCOUNT + 1; CARDCOUNT := R0;
IF LISTFLAG THEN
  BEGIN
    CVD(R0,CONWORK); UNPK(3,7,WBUF(45),CONWORK);
    SETZONE(WBUF(48)); MVC(79,WBUF(52),CBUF);
    R0 := @WBUF; WRITE; MVC(131,WBUF,BLANK);
  END;
CP := 0; R4 := SAVE4;
END;

PROCEDURE SCAN(R4);
  BEGIN INTEGER SAVE4, S1, S2;
  SAVE4 := R4; R0 := 0; NUMBERVALUE := R0;
  WHILE TRUE DO
    BEGIN
      IF CP > TEXTLIMIT THEN GETCARD ELSE
        BEGIN COMMENT BRANCH ON NEXT CHARACTER IN CBUF;
          IC(R1,CBUF(CP)); R1 := R1 AND MASK;
          IC(R2,CHARTYPE(R1)); R2 := R2 AND MASK;
          IF R2 < 10 THEN CASE R2 OF
            BEGIN
              COMMENT CASE 1: ILLEGAL CHARACTER;
              BEGIN
                IF ENDIT THEN
                  BEGIN
                    R1 := 1; TOKEN := R1; GOTO L1;
                  END;
                XR := 2; ERROR;
              END;
              COMMENT CASE 2: BLANK;
              BEGIN COMMENT SKIP OVER BLANKS;
                CP := CP + 1;
                IF CP <= TEXTLIMIT THEN
                  BEGIN
                    IC(R2,CBUF(CP)); R2 := R2 AND MASK;
                    WHILE R2 = 64 AND CP < TEXTLIMIT DO
                      BEGIN
                        CP := CP + 1; IC(R2,CBUF(CP));
                      END;
                  END;
                IF CP ^= TEXTLIMIT THEN CP := CP - 1;
              END;
              COMMENT CASES 3 & 4 NOT USED;
              NULL;
              NULL;
              COMMENT CASE 5: A LETTER;
              BEGIN
                S1 := CP; R1 := 0; S2 := R1;
                F45 := 0L; BCD := F45;
                WHILE TRUE DO COMMENT UNTIL END OF IDENT;
                  BEGIN
                    FOR CP := CP STEP 1 UNTIL TEXTLIMIT DO
                      BEGIN
                        IC(R1,CBUF(CP)); R1 := R1 AND MASK;
                        IC(R2,NOTLETTERORDIGIT(R1));
                        R2 := R2 AND MASK;
                        IF R2 = 1 THEN
                          BEGIN COMMENT END OF IDENTIFIER;
                            IF CP > S1 THEN R2 := CP - S1
                              ELSE R2 := TEXTLIMIT - S1 + CP
                              + 1; R1 := 0; S1 := R1;
                            CP := CP - 1;
                          COMMENT R2 IS LENGTH OF IDENT;
                            R5 := NUMTERMINALS SHLL 3;
                            IF R2 <= RESERVEDLIMIT THEN
                              FOR R1 := 8 STEP 8 UNTIL R5 DO

```



```

        BEGIN
            F67 := V(R1);
            IF F67 = BCD THEN
                BEGIN
                    R1 := R1 SHRL 3;
                    TOKEN := R1; GOTO L1;
                END;
            END;
        COMMENT MUST BE <IDENT>;
        R1 := IDENT; TOKEN := R1;
        GOTO L1;
    END ELSE COMMENT LETTER OR DIGIT;
    BEGIN
        R2 := S2 + 1; S2 := R2;
        IF R2 <= 4 THEN
            BEGIN
                R4 := R4 - R4; BCDHIGH := R4;
                IC(R4,CBUF(CP));
                R4 := R4 AND MASK;
                R5 := BCDLOW SHLL 8 OR R4;
                BCDLOW := R5;
            END ELSE
            BEGIN
                IC(R4,CBUF(CP));
                R4 := R4 AND MASK;
                R5 := BCDHIGH SHLL 8 OR R4;
                BCDHIGH := R5;
            END;
        END;
    END;
    COMMENT END OF CARD;
    GETCARD;
    END;
END;
COMMENT CASE 6: DIGIT;
BEGIN
    R1 := NUMBER; TOKEN := R1; R1 := R1 - R1;
    WHILE TRUE DO COMMENT UNTIL GOTO L1;
    BEGIN
        FOR CP := CP STEP 1 UNTIL TEXTLIMIT DO
            BEGIN
                IC(R1,CBUF(CP));
                IF R1 < 240 THEN GOTO L1;
                R3 := NUMERVALUE * 10 + R1 - 240;
            END;
        GETCARD;
    END;
    END;
    COMMENT CASE 7: /;
    BEGIN
        R1 := DIVIDE; TOKEN := R1; CP := CP + 1;
        GOTO L1;
    END;
    COMMENT CASE 8: SPECIAL CHARACTER;
    BEGIN
        R1 := R1 SHLL 2; R2 := TX(R1);
        TOKEN := R2; CP := CP + 1; GOTO L1;
    END;
    COMMENT CASE 9: END OF FILE MARK,".";
    BEGIN
        R1 := 1; TOKEN := R1; GOTO L1;
    END;
END; COMMENT END OF CASE ON CHARTYPE;
CP := CP + 1;
END;
END;
L1: R4 := SAVE4;
END;

PROCEDURE PRINTIME(R6); NULL;
COMMENT THIS PROCEDURE SHOULD GET TIME OF DAY AND INSERT
        INTO WBUF(18)-WBUF(22), FORMAT "13:37";

```



```

PROCEDURE PRINTDATE(R6); NULL;
COMMENT THIS PROCEDURE SHOULD GET TODAY'S DATE AND INSERT
        INTO WBUF(9)-WBUF(14), FORMAT "72.153";

PROCEDURE INITIALIZE(R4);
COMMENT THIS PROCEDURE SETS VARIABLES TO BE USED IN THE
        PROGRAM. IT SHOULD ALSO SAVE THE CURRENT TIME OF
        DAY TO BE USED IN PROCEDURE SUMMARIZE;
BEGIN INTEGER SAVE4;
        SAVE4 := R4; MVC(131,WBUF,BLANK); R0 := @WBUF;
        MVC(34,WBUF,"REPLACE THIS HEADING WITH ONE OF");
        MVC(32,WBUF(36),"YOUR OWN -- VERSION OF JUNE 1972.");
        WRITE; MVC(68,WBUF,BLANK); WRITE; WRITE; PRINTDATE;
        PRINTIME; MVC(8,WBUF,"TODAY IS ");
        MVC(0,WBUF(16),"@");
        WRITE; MVC(131,WBUF,BLANK); WRITE; WRITE;
        R5 := NUMTERMINALS;
        COMMENT SEARCH THE TERMINAL SYMBOLS FOR "-|_",
                "<NUMBER>", "<IDENT>", AND "/";
        FOR R1 := 1 STEP 1 UNTIL R5 DO
        BEGIN
                R2 := R1 SHLL 3; F45 := V(R2);
                IF F45 = #6D4F6DL THEN EOFILF := R1 ELSE
                IF F45 = #4CD5E4D4C2C5D96EL THEN NUMBER := R1 ELSE
                IF F45 = #4CC9C4C5D5E36EL THEN IDENT := R1 ELSE
                IF F45 = #61L THEN DIVIDE := R1;
        END;
        F45 := #C5D6C6L; COMMENT SAME AS "EOF"; V8 := F45;
        R3 := R3 - R3; IC(R3," "); R4 := @CHARTYPE(R3);
        MVI(2,B4); IC(R3,"."); R4 := @CHARTYPE(R3); MVI(9,B4);

COMMENT IF LETTERS ARE TO BE RECOGNIZED, INSERT THE
        FOLLOWING LOOP. (NOTE, SEMICOLONS ARE REMOVED TO
        AVOID ENDING THIS COMMENT).
        FOR R1 := 0 STEP 1 UNTIL 29 DO
        COMMENT LENGTH OF ALPHABET IS 30
        BEGIN
                R3 := R3 - R3 IC(R3,ALPHABET(R1))
                R4 := @CHARTYPE(R3) MVI(5,B4)
                R4 := @NOTLETTERORDIGIT(R3) MVI(0,B4)
                R3 := R3 SHLL 2 TX(R3) := R1
        END;

        R3 := R3 - R3;

COMMENT SAME FOR NUMBERS:
        FOR R1 := 0 STEP 1 UNTIL 9 DO
        BEGIN
                IC(R3,NUMS(R1)) R4 := @CHARTYPE(R3) MVI(6,B4)
                R4 := @NOTLETTERORDIGIT(R3) MVI(0,B4)
        END;

        FOR R1 := 2 STEP 1 UNTIL R5 DO
        BEGIN
                R2 := R1 SHLL 3; F45 := V(R2);
                IF F45 < #FFL THEN
                BEGIN COMMENT SPECIAL CHARACTER;
                        R3 := R3 - R3; R4 := R1 SHLL 3 + 7;
                        IC(R3,V(R4)); R4 := @CHARTYPE(R3); MVI(8,B4);
                        R3 := R3 SHLL 2; TX(R3) := R1;
                END ELSE COMMENT SET RESERVEDLIMIT;
                BEGIN
                        IF F45 < #FFFFFL THEN R3 := 2 ELSE
                        IF F45 < #FFFFFFFL THEN R3 := 3 ELSE
                        IF F45 < #FFFFFFFFFL THEN R3 := 4 ELSE
                        IF F45 < #FFFFFFFFFFFFFL THEN R3 := 5 ELSE
                        IF F45 < #FFFFFFFFFFFFFFFFFL THEN R3 := 6 ELSE
                        IF F45 < #FFFFFFFFFFFFFFFFFFFFFL THEN R3 := 7 ELSE
                        R3 := 8;
                        IF R3 > RESERVEDLIMIT THEN
                        BEGIN

```



```

        R4 := R1 - R3 + 8; R5 := @V(R4); CLI("<",B5);
        IF  $\neg$  THEN RESERVEDLIMIT := R3;
    END;
END;
END;
CP := TEXTLIMIT + 1; MVC(79,CBUF,BLANK); R1 := 0;
SP := R1; SET(LISTFLAG); RESET(ENDIT); R4 := SAVE4;
END; COMMENT END INITIALIZE;

PROCEDURE EMIT(R4); NULL;
COMMENT THIS PROCEDURE HAS THE RESPONSIBILITY OF SETTING
        THE NEXT ELEMENT OF THE CODE ARRAY TO THE OPCODE
        DETERMINED BY PROCEDURE SYNTHESIZE;

PROCEDURE LOOKUP(R4); NULL;
COMMENT THIS PROCEDURE LOOKS UP A NAME IN THE SYMBOL TABLE
        AND ENTERS IT IF NOT THERE;

PROCEDURE SYNTHESIZE(R4); NULL;
COMMENT THIS PROCEDURE IS RESPONSIBLE FOR THE SEMANTICS OF
        THE COMPILER. IT TAKES THE FORM OF A LARGE CASE
        STATEMENT ON GLOBAL VARIABLE PRODNUM. ARRIVE HERE
        FROM THE CASE STATEMENT IN PROCEDURE ANALYZE;

PROCEDURE PRINTSUMMARY(R4);
COMMENT THIS PROCEDURE SHOULD SUBTRACT CURRENT TIME OF DAY
        WITH THAT SAVED IN PROCEDURE INITIALIZE AND STORE
        THE RESULT IN WBUF STARTING IN COLUMN 19;
BEGIN INTEGER SAVE4;
    SAVE4 := R4;
    MVC(17,WBUF,"TIME IN EXECUTION:"); R0 := @WBUF; WRITE;
    MVC(17,WBUF,BLANK);
    R4 := SAVE4;
END;

PROCEDURE ANALYZE(R4);
BEGIN INTEGER SAVE4, NEXTSYMBOL;
    ARRAY 75 INTEGER STATESTACK = 75(0);
    INTEGER STATENUM = 0, LASYMBOL = 0;

    PROCEDURE PUSHANDREAD(R4);
        BEGIN INTEGER SAVE4;
            SAVE4 := R4; R1 := SP;
            IF R1 < 75 THEN
                BEGIN
                    R1 := R1 + 1; SP := R1;
                END ELSE
                BEGIN
                    XR := 3; ERROR; GOTO EXIT;
                END;
            R1 := TOKEN; NEXTSYMBOL := R1;
            COMMENT SET VAR(SP) TO BCD AND VAL(SP) TO
                    NUMBERTOVALUE;
            SCAN; R4 := SAVE4;
        END; COMMENT END PUSHANDREAD;

    INTEGER CYCLECNT = 0; SAVE4 := R4;
    WHILE TRUE DO
        BEGIN
            R1 := CYCLECNT + 1; CYCLECNT := R1;
            R1 := SP SHLL 2; R2 := STATESTACK(R1) SHRL 8 + 1;
            CASE R2 OF
                BEGIN
                    COMMENT CASE 1, READ VIA LINEAR SEARCH;
                    BEGIN
                        PUSHANDREAD; R1 := STATENUM SHLL 1;
                        R2 := READSTART(R1); R1 := R1 SHRL 1;
                        IC(R3,RDNUM(R1)); R3 := R3 AND MASK;
                        R3 := R3 + R2; R4 := NEXTSYMBOL;
                        IC(R5,SYMLIST(R2)); R5 := R5 AND MASK;
                        WHILE R4  $\neg$  R5 AND R2 < R3 DO
                            BEGIN

```



```

        R2 := R2 + 1; IC(R5, SYMLIST(R2));
    END;
    R2 := R2 SHLL 1;
    R1 := STATELIST(R2) AND MASKFFFF;
    R2 := SP SHLL 2; STATESTACK(R2) := R1;
    R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT CASE 2, READ VIA AN ARRAY ACCESS;
BEGIN
    PUSHANDREAD; R1 := STATENUM SHLL 1;
    R2 := READSTART(R1) + NEXTSYMBOL SHLL 1;
    R1 := STATELIST(R2) AND MASKFFFF;
    R2 := SP SHLL 2; STATESTACK(R2) := R1;
    R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT CASE 3, REDUCE;
BEGIN
    R1 := STATENUM; PRODNUM := R1; SYNTHESIZE;
    R1 := STATENUM; IC(R2, NUMTOPOP(R1));
    R2 := R2 AND MASK; R3 := SP - R2; SP := R3;
    R1 := R1 SHLL 1; R2 := REDUCESUCC(R1);
    R1 := SP SHLL 2; STATESTACK(R1) := R2;
    R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT CASE 4, LOOK AHEAD (ORDINARY);
BEGIN
    R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
    R3 := R1 AND MASK7; R4 := 7 - R3;
    R5 := STATENUM SHLL 2 + R2;
    IC(R1, LATABLE(R5));
    R1 := R1 AND MASK SHRL R4 AND MASK1;
    IF R1 = 1 THEN
        BEGIN
            R1 := STATENUM SHLL 1;
            R2 := SUCCSTATE(R1);
        END ELSE
        BEGIN
            R1 := STATENUM SHLL 1;
            R2 := FAILSTATE(R1);
        END;
    R2 := R2 AND MASKFFFF; R1 := SP SHLL 2;
    STATESTACK(R1) := R2; R2 := R2 AND MASK;
    STATENUM := R2;
END;
COMMENT CASE 5, LOOK AHEAD (FOR A PRODUCTION
WITH AN EMPTY RIGHT PART);
BEGIN
    R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
    R3 := R1 AND MASK7; R4 := 7 - R3;
    R5 := STATENUM SHLL 2 + R2;
    IC(R1, LATABLE(R5));
    R1 := R1 AND MASK SHRL R4 AND MASK1;
    IF R1 = 1 THEN
        BEGIN
            R1 := SP SHLL 2;
            R2 := STATESTACK(R1) SHRL 8;
            WHILE R2 = 3 OR R2 = 4 DO
                BEGIN
                    R3 := STATESTACK(R1) AND MASK SHLL 1;
                    R4 := FAILSTATE(R3);
                    STATESTACK(R1) := R4;
                    R2 := R4 SHRL 8;
                END;
            R1 := SP + 1; SP := R1;
            R1 := STATENUM SHLL 1;
            R2 := SUCCSTATE(R1);
        END ELSE
        BEGIN
            R1 := STATENUM SHLL 1;
            R2 := FAILSTATE(R1);
        END;
    R1 := SP SHLL 2; STATESTACK(R1) := R2;

```



```

    R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT CASE 6, LOOK BACK;
BEGIN
    R1 := SP - 1 SHLL 2; R2 := STATENUM;
    IC(R3,LBSTART(R2)); R3 := R3 AND MASK;
    IC(R4,LBNUM(R2)); R4 := R4 AND MASK + R3;
    R3 := R3 SHLL 1; R5 := LBSTATE(R3);
    R6 := STATESTACK(R1); R3 := R3 SHRL 1;
    WHILE R6 ≠ R5 AND R3 < R4 DO
    BEGIN
        R3 := R3 + 1 SHLL 1; R5 := LBSTATE(R3);
        R3 := R3 SHRL 1;
    END;
    R3 := R3 SHLL 1; R1 := RESUMESTATE(R3);
    R2 := SP SHLL 2; STATESTACK(R2) := R1;
    R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT CASE 7, ERROR;
BEGIN INTEGER PREVERRCYCLE = #FFFFFFFF;
    R1 := CYCLECNT - 2; CYCLECNT := R1;
    IF R1 ≠ PREVERRCYCLE THEN
    BEGIN
        PREVERRCYCLE := R1; R1 := SP - 1 SHLL 2;
        R2 := STATESTACK(R1);
        IF R2 < 512 THEN
        BEGIN
            R2 := R2 AND MASK;
            IC(R3,SYMBEFORELREAD(R2));
        END ELSE
        BEGIN
            R2 := STATENUM; IC(R3,SYMBEFORELA(R2));
        END;
        R3 := R3 AND MASK SHLL 3;
        F45 := V(R3); VR3 := F45;
        R1 := VR3LOW OR BLANKMASK; VR3LOW := R1;
        R1 := VR3HI OR BLANKMASK; VR3HI := R1;
        MVC(7,WBUF(32),VR3); R3 := STATENUM;
        IF R3 = 255 THEN R3 := NEXTSYMBOL
        ELSE R3 := LASYMBOL; R3 := R3 SHLL 3;
        F45 := V(R3); VR3 := F45;
        R1 := VR3LOW OR BLANKMASK; VR3LOW := R1;
        R1 := VR3HI OR BLANKMASK; VR3HI := R1;
        MVC(7,WBUF(41),VR3); XR := 4; ERROR;
        MVC(17,WBUF,"PARTIAL PARSE IS: ");
        R0 := @WBUF; WRITE; MVC(17,WBUF,BLANK);
        R2 := SP - 1 SHLL 2;
        FOR R1 := 8 STEP 4 UNTIL R2 DO
        BEGIN
            R3 := STATESTACK(R1);
            IF R3 < 512 THEN
            BEGIN
                R3 := R3 AND MASK;
                IC(R4,SYMBEFORELREAD(R3));
            END ELSE
            BEGIN
                R3 := R3 AND MASK;
                IC(R4,SYMBEFORELA(R3));
            END;
            R4 := R4 AND MASK SHLL 3;
            F45 := V(R4); VR3 := F45;
            R1 := VR3LOW OR BLANKMASK;
            VR3LOW := R1;
            R1 := VR3HI OR BLANKMASK; VR3HI := R1;
            MVC(7,WBUF(4),VR3); WRITE;
            MVC(7,WBUF(4),BLANK);
        END;
    END;
    R1 := NEXTSYMBOL;
    IF R1 = 1 THEN
    BEGIN
        XR := 5; ERROR;
    END;

```



```

END ELSE
BEGIN
    R1 := R1 SHLL 3;
    MVC(16,WBUF,"THE INPUT SYMBOL,");
    F45 := V(R1); VR3 := F45;
    R1 := VR3LOW OR BLANKMASK; VR3LOW := R1;
    R1 := VR3HI OR BLANKMASK; VR3HI := R1;
    MVC(7,WBUF(18),VR3);
    MVC(17,WBUF(26)," WILL BE IGNORED.");
    RO := @WBUF; WRITE; MVC(48,WBUF,BLANK);
END;
R1 := STATENUM;
IF R1 = 255 THEN
BEGIN
    COMMENT ERROR OCCURRED IN A READ STATE;
    R1 := SP - 1; SP := R1; R1 := R1 SHLL 2;
    R2 := STATESTACK(R1) AND MASK;
    STATENUM := R2;
END ELSE
BEGIN
    COMMENT ERROR OCCURRED IN A LOOK-AHEAD STATE;
    SCAN; COMMENT SKIP THE NEXT SYMBOL;
    R1 := R1 SHLL 1;
    R2 := SUCCSTATE(R1) AND MASKFF00;
    R3 := SUCCSTATE(R1) AND MASK;
    IC(R4,NUMTOPOP(R3)); R4 := R4 AND MASK;
    IF R2 = 512 AND R4 = 255 THEN
        R1 := STATENUM OR #00000300
    ELSE R1 := STATENUM OR #00000400;
    R2 := SP SHLL 2; STATESTACK(R2) := R1;
END;
END; COMMENT END OF CASE 7;
BEGIN COMMENT EXIT;
    R1 := 1; TOKEN := R1; R1 := 0;
    SP := R1; STATESTACK(R1) := R1;
    STATENUM := R1; GOTO XXX;
END;
END; COMMENT END OF CASE(STATETYPE);
END;
XXX: R4 := SAVE4;
END;

PROCEDURE MAIN(R4);
BEGIN INTEGER SAVE4;
    SAVE4 := R4;
    INITIALIZE;
    ANALYZE;
    PRINTSUMMARY;
    R4 := SAVE4;
END;

MAIN;
EXIT;
MVC(17,WBUF,"END OF COMPILATION"); RO := @WBUF; WRITE;
R2 := MEM(R13+4); RO := 0;
MEM(R2+16) := RO; COMMENT SET RETURN CODE;
END.

```


APPENDIX B

SYNTAX ANALYZER OUTPUT

```

/* CARDS PUNCHED BY THE SLR(1) GRAMMAR ANALYSER. */
DECLARE #_TERMINALS LITERALLY '6',
#_NTS LITERALLY '4',
#_SYMS LITERALLY '10';

DECLARE V(#_SYMS) CHARACTER INITIAL ('ERROR_SYMBOL', '_|_', '++', '*', 'X',
', 'Y', 'Z', '<G>', '<E>', '<I>', '<P>');

/* THE DPDA HAS 8 READ STATES. */
DECLARE READ_START(7) BIT(16) INITIAL (/# 0 #/ 0, 2, 9, 11, 13, 15, 22, 29
);

DECLARE RD_#(7) BIT(8) INITIAL (/# 0 #/ 1, 7, 1, 1, 1, 7, 7, 1);

DECLARE SYM_LIST(30) BIT(8) INITIAL (/# 0 #/ 1, 0, 0, 1, 2, 3, 4, 5, 6, 1
, /# 10 #/ 0, 2, 0, 3, 0, 0, 1, 2, 3, 4, /# 20 #/ 5, 6, 0, 1, 2, 3, 4
, 5, 6, 3, /# 30 #/ 0);

DECLARE STATE_LIST(30) BIT(16) INITIAL (/# 0 #/ 257, 1791, 1791, 1791
, 1791, 1791, 518, 519, 520, 521, /# 10 #/ 1791, 261, 1791, 262, 1791
, 1791, 1791, 1791, 1791, 518, /# 20 #/ 519, 520, 1791, 1791, 1791
, 1791, 518, 519, 520, 262, /# 30 #/ 1791);

/* THE DPDA HAS 9 REDUCE STATES. */
DECLARE #_TO_POP(9) BIT(8) INITIAL (/# 0 #/ 0, 0, 2, 0, 2, 0, 0, 0, 2);

DECLARE REDUCE_SUCC(9) BIT(16) INITIAL (/# 0 #/ 0, 2, 768, 768, 1280, 1280
, 1281, 1281, 1281, 1792);

/* THE DPDA HAS 3 LOOK-AHEAD STATES. */
DECLARE LA_SYM_#(2) BIT(8) INITIAL (/# 0 #/ 0, 0, 0);

DECLARE SUCC_STATE(2) BIT(16) INITIAL (/# 0 #/ 513, 514, 515);

DECLARE FAIL_STATE(2) BIT(16) INITIAL (/# 0 #/ 3, 4, 7);

```



```

DECLARE LA_TABLE(2) BIT(33) INITIAL (
  "(1)01000000000000000000000000000000" /* <G> */,
  "(1)01000000000000000000000000000000" /* <E> */,
  "(1)01100000000000000000000000000000" /* <E> */);

/* THE OPDA HAS 2 LOOK-BACK STATES. */
DECLARE LB_START(1) BIT(8) INITIAL (/ 0 0, 2);
DECLARE LB_#(1) BIT(8) INITIAL (/ 0 0, 1, 1);
DECLARE LB_STATE(3) BIT(16) INITIAL (/ 0 0, 257, 0, 262, 0);
DECLARE RESUME_STATE(3) BIT(16) INITIAL (/ 0 0, 769, 770, 517, 516);

/* THE SYMBOLS ACCESSING THE STATES. */
DECLARE SYM_BEFORE_READ(7) BIT(8) INITIAL (/ 0 0, 1, 7, 8, 9, 2, 3, 9
);
DECLARE SYM_BEFORE_LA(2) BIT(8) INITIAL (/ 0 0, 8, 9, 9);

/* END OF CARDS PUNCHED BY THE SLR(1) GRAMMAR ANALYSER. */

```


APPENDIX C

OS/360 OPERATING SYSTEM INTERFACE

ICTL 1,71,18
SPACE

```

*
*****
*
*      PLIO -- THE RUNTIME OS INTERFACE FOR PROTOCOM
*
*****
*
SPACE
MACRO
&EP ENTER
ENTRY &EP
USING &EP,15
&EP STM 12,2,SAVE SAVE REGISTERS
L 12,=A($PL360IO) ESTABLISH ADDRESSING
USING $PL360IO,12
DROP 15
MEND
SPACE
MACRO
EXIT
LM 12,2,SAVE RESTORE REGISTERS
BR 14
DROP 12
MEND
SPACE 2
$PL360IO CSECT
SPACE
LINELEN EQU 132 PRINTER LINE LENGTH
LINESMAX EQU 60 PRINTER LINES/PAGE
PRINT NOGEN
SPACE
* GLOBAL PROCEDURE READ(R14)
* (R0) = BUFFER ADDRESS
* (R13) = SAVE AREA ADDRESS
* (R14) = RETURN ADDRESS
READ ENTER
TM SYSIN+DOPEN,OPENMASK TEST FOR OPEN DCB
BO READ1
LR 2,0
OPEN (SYSIN,(INPUT)) ISSUE OPEN
LR 0,2
READ1 CLI EOF,X'FF' TEST FOR PREVIOUS
* END-OF-FILE
BE ERRPROC
GET SYSIN,(0) GET CARD
READ2 CLI EOF,0 SET CONDITION CODE
EXIT
SPACE
* EOD EXIT ROUTINE
ENDRDR USING $PL360IO,12
MVI EOF,X'FF' EODAD EXIT
B READ2
DROP 12
SPACE 2
* GLOBAL PROCEDURE WRITE(R14)
* (R0) = BUFFER ADDRESS
* (R13) = SAVE AREA ADDRESS
* (R14) = RETURN ADDRESS
WRITE ENTER
LR 2,0
TM SYSOUT+DOPEN,OPENMASK
BO WRITE1

```


	OPEN	(SYSOUT,(OUTPUT))	
WRITE1	LR	1,0	STORE BUFFER ADDRESS IN R1
	MVC	ABUFF+1(132),0(1)	MOVE TO OUR BUFFER
	MVC	ABUFF(1),CARRCONT	CARRIAGE CONTROL TO ABUFF
	LA	0,ABUFF	
	PUT	SYSOUT,(0)	GET NEXT BUFFER
*			ADDRESS IN R1
	MVC	0(1,1),CARRCONT	SET CONTROL CHARACTER
	CLI	CARRCONT,C'1'	
	BNE	WRITE2	
	MVI	LINECNT,0	RESET LINE COUNT
WRITE2	MVI	CARRCONT,C'1'	
	MVC	1(LINELEN,1),0(2)	TRANSFER BUFFER
	IC	2,LINECNT	INCREMENT LINE COUNT
	LA	2,1(,2)	
	STC	2,LINECNT	
	CLI	LINECNT,LINESMAX	TEST FOR FULL PAGE
	BL	WRITE3	
	MVI	CARRCONT,C'1'	SET SKIP
WRITE3	LM	12,2,SAVE	
	BR	14	
	DROP	12	
	SPACE	2	
*	GLOBAL PROCEDURE	PAGE(R14)	
*	(R14)	= RETURN ADDRESS	
	ENTRY	PAGE	
	USING	PAGE,15	
PAGE	MVI	CARRCONT,C'1'	
	BR	14	
	SPACE	2	
*	GLOBAL PROCEDURE	PUNCH(R14)	
*	(R0)	= BUFFER ADDRESS	
*	(R13)	= SAVE AREA ADDRESS	
*	(R14)	= RETURN ADDRESS	
PUNCH	ENTER		
	TM	SYPUNCH+DOPEN,OPENMASK	
	BO	PUNCH1	
	LR	2,0	
	OPEN	(SYPUNCH,(OUTPUT))	
	LR	0,2	
PUNCH1	PUT	SYPUNCH,(0)	PUT CARD IMAGE
	EXIT		
	SPACE	2	
SYSOUT	DCB	DSORG=PS,MACRF=PL,DDNAME=SYSPRINT,DEVD=DA,	
		RECFM=FBA,LRECL=LINELEN+1,BFTEK=S,EROPT=ABE	
SYSIN	DCB	DSORG=PS,MACRF=GM,DDNAME=SYSIN,DEVD=DA,	
		RECFM=FB,LRECL=80,BFTEK=S,EROPT=ABE,	
		EODAD=ENDRDR	
SYPUNCH	DCB	DSORG=PS,MACRF=PM,DDNAME=SYPUNCH,DEVD=DA,	
		RECFM=FB,LRECL=80,BFTEK=S,EROPT=ABE	
SAVE	DS	7F	
ERRPROC	DC	H'0'	
EOF	DC	X'00'	
LINECNT	DC	X'00'	PRINTER LINE COUNT
CARRCONT	DC	C'1'	CARRIAGE CONTROL
	SPACE		
ABUFF	DS	CL133	
	SPACE		
	DCBD		
\$PL360IO	CSECT		
DOPEN	EQU	DCBOFLGS-IHADCB	BYTE SET BY OPEN
OPENMASK	EQU	X'10'	
	END		

APPENDIX D

JOB CONTROL LANGUAGE

```

//BLANCH#2      JOB      (0938,0584NT,CS12),'BLANCHARD',TIME=(,5)
//JCBLIB        DD      DSN=SO938.PLLIB,UNIT=2314,VOL=SER=MARY,DISP=SHR
//COMP          EXEC     PGM=PL360,REGION=100K
//SYSPRINT      DD      SYSOUT=A,SPACE=(CYL,(1,1)),BLKSIZE=798)
//              DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
//SYSGO         DD      UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE),DISP=(,PASS),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN         DD      *
//              NULL; COMMENT NULL PL360 PROGRAM;
//LINK          EXEC     PGM=IEWL,PARM=MAP,LIST',REGION=100K,COND=(0,NE,COMP)
//SYSUT1        DD      UNIT=SYSDA,SPACE=(TRK,(20,10))
//SYSLSMOD      DD      DSN=ET(PROGRAM),UNIT=SYSDA,DISP=(,PASS),
//              SPACE=(CYL,(1,1),RLSE)
//SYSLIB        DD      DSN=SO938.PLLIB,UNIT=2314,VOL=SER=MARY,DISP=SHR
//SYSPRINT      DD      SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
//              SPACE=(CYL,(1,1))
//SYSLIN        DD      DSN=*.COMP.SYSGO,DISP=(OLD,DELETE)
//GO            EXEC     PGM=*.LINK.SYSLMOD,REGION=58K,COND=(4,LT,LINK)
//SYSPRINT      DD      SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798),
//              SPACE=(CYL,(1,1))
//SYSIN         DD      *
DATA CARD.

```


BIBLIOGRAPHY

1. Wirth, N. PL360, A Programming Language for the 360 Computers. J. ACM 15 (1968), 37-74.
2. DeRemer, F. L. Simple LR(k) Grammars. Comm. ACM 14 (1971), 453-460.
3. Feldman, J., and Gries, D. Translator Writing Systems. Comm. ACM 11 (1968), 77-113.
4. O'Neil, J. T., Jr. META PI, An On-line Interactive Compiler-compiler. Proc. AFIPS 1968 FJCC, Vol. 33, AFIPS Press, Montvale, N. J., p. 201.
5. McKeeman, W. M., Horning, J. J., and Wortman, D. B. A Compiler Generator, Prentice-Hall, Englewood Cliffs, N. J., 1970.
6. Schorre, D. V. META II, A Syntax-oriented Compiler Writing Language. Proc. ACM, 19th Nat. Conf., 1964.
7. Wirth, N. and Weber, H. EULER - A Generalization of ALGOL, and its Formal Definition. Comm. ACM 9 (1966), 13-25, 89-99.
8. OS/360 PL360 Compiler. IBM Contributed Library (Type IV) Program Number 360D-03.2.011.
9. Malcolm, M. A. PL360 (Revised), A Programming Language for the IBM 360, Stanford Univ., May 1971.
10. DeRemer, F. L. Simple LR(k) Grammars: Definition and Implementation. CEP Rep. 2, 4 (Sept. 1970), U. of Calif., Santa Cruz.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Asst Professor G. A. Kildall, Code 53Kd Department of Mathematics Naval Postgraduate School Monterey, California	1
4. Asst Professor G. E. Heidorn, Code 55Hd Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5. LT R. C. Blanchard, USN 838 Huntington Mt. Clemens, Michigan 48043	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)		2a. REPORT SECURITY CLASSIFICATION	
Naval Postgraduate School Monterey, California 93940		Unclassified	
		2b. GROUP	
3. REPORT TITLE			
Steps Toward a PL360-Based Compiler Generator for the IBM 360 Computer			
4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)			
Master's Thesis; June 1972			
5. AUTHOR(S) (First name, middle initial, last name)			
Robert C. Blanchard			
6. REPORT DATE		7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1972		38	10
8a. CONTRACT OR GRANT NO.		9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT			
Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
		Naval Postgraduate School Monterey, California 93940	
13. ABSTRACT			
<p>The documentation of a complete proto-compiler consisting of a syntax checker and an OS/360 operating system interface for the IBM System/360 computers is presented. The system constitutes the foundation of a translator writing system based on the language PL360 and on the SLR(k) parsing algorithm. PL360 provides all the facilities of a symbolic machine language but displays an ALGOL-like structure for improved readability and programming ease. SLR(k) parsers have been shown to be superior to those constructed using precedence techniques with regard to the class of acceptable grammars and speed of operation.</p>			

14. KEY WORDS

Compiler Generator

PL360

Proto-Compiler

Syntax Checker

LR(k) Parser

SLR(k) Parser

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

14 FEB 73

20192

Thesis

B5485

Blanchard

c.1

Steps toward a
PL360-based compiler
generator for the IBM
360 computer.

134827

14 FEB 73

20192

Thesis

B5485

Blanchard

c.1

Steps toward a
PL360-based compiler
generator for the IBM
360 computer.

134827

thesB5485

Steps toward a PL 360-based compiler gen



3 2768 002 13536 0

DUDLEY KNOX LIBRARY